AIAA 2002-4673

Autoreturn Function for a Remotely Piloted Vehicle

J. D. McMinn and E. Bruce Jackson
NASA Langley Research Center
Hampton, Virginia

Guidance, Navigation & Control Conference

5-8 August 2002
Monterey, California

AUTORETURN FUNCTION FOR A REMOTELY PILOTED VEHICLE

J. D. McMinn[*] and E. Bruce Jackson[*]
NASA Langley Research Center
Hampton, Virginia

Abstract

An algorithm to maneuver an air vehicle to intercept and follow a pre-planned path while remaining within an arbitrary, closed boundary is outlined. The immediate application is for an autonomous lost-link return-to-runway function for a remotely piloted vehicle being developed by NASA, but other applications are hypothesized. Results of implementation in a flight simulator are given.

Nomenclature

D	distance between turn circle centers
g	gravitational acceleration
j	turn direction indicator = +1 for a turn that begins with a right turn = –1 for a turn sequence begins with a left turn
k	intercept trajectory style = –1 for a turn that is first left then right or vice versa = +1 for right then right or left then left
L	linear measurement (figure specific)
n	number of points defining boundary
R	turn radius
v	velocity
x	boundary coordinate (+ East)
y	boundary coordinate (+ North)
β	angular measurement (figure specific)
γ	angular measurement (figure specific)
ϕ	bank angle
ψ	course from one turn circle to next
χ	track angle

subscripts

a/c	aircraft
cmd	commanded value
i	boundary index number
nav	from navigation module
nom	nominal value

Introduction

Remotely-piloted and unmanned air vehicles are becoming increasingly common, especially in the development process of new aerospace vehicles. During the development of a remotely-piloted vehicle at NASA, an autonomous return to the runway in the event of the loss of command uplink was required. Since the vehicle was to be operated within a restricted flight test range it was imperative that the vehicle avoid penetrating the range boundary in the process of maneuvering to intercept and follow the desired preplanned flight path back to the runway.

This paper describes the elements of an autoreturn algorithm meant to meet this requirement. This algorithm has been realized and tested in a real-time flight simulation. The scope of this paper includes the elements unique to the autoreturn function; it does not describe the underlying control laws or autopilot (heading- and altitude-hold) subsystems.

Applications of this algorithm include any autonomously maneuvering vehicle that must plan in real time to avoid specific areas of airspace while performing a preplanned mission.

Autoreturn Requirements

Because the air vehicle to which this algorithm applies will be unmanned (normally remotely-piloted), the Air Force Flight Test Center and NASA's Dryden Flight Research Center, where the vehicle will be flown, require a Flight Termination System (FTS) that will effectively destroy the vehicle if control is lost and the vehicle has penetrated the boundaries surrounding the designated testing area at Edwards Air Force Base. To avoid operation of the FTS in the case of loss of command uplink, the vehicle is to be equipped with a boundary detection and avoidance function. With additional software, some well chosen waypoints, and an accurate definition of the boundaries, an autoreturn function can be developed to guide the vehicle back to

[*] Senior Research Engineer, Dynamics and Control Branch, Airborne Systems Competency; Member, AIAA

the launch point without a boundary excursion, and to execute an automatic landing.

The autoreturn function provides four-dimensional guidance to bring the vehicle back to the runway for the automatic landing. The autoreturn function works in conjunction with a vehicle-specific autopilot that provides heading command and flight path or altitude command features.

Autoreturn Design

The nominal autoreturn path is predetermined and hard-coded into the vehicle's flight control system. Since the vehicle's position at the time of loss-of-link is unpredictable, the autoreturn function must provide guidance to steer the vehicle from its present position onto the preplanned path, without exceeding vehicle flight envelope and range boundaries.

In addition to this requirement, the vehicle must detect an out-of-nominal approach path to the runway and initiate and perform a go-around maneuver when necessary.

The connection between the autoreturn function and the autopilot is a guidance law that takes errors in the vehicle's path and steers it back to the desired path.

Nominal Flight Path

As shown in figure 1, the nominal trajectory consists of an ordered set of waypoints that lead from the test area to the runway landing pattern. The waypoints are defined in terms of latitude, longitude, altitude, and airspeed.

The flight path designer must also specify a nominal bank angle to use in steering, which should not exceed the maximum allowable bank angle. Combined with scheduled airspeed, this bank angle determines the turning radius (in still air) of the vehicle at each waypoint.

Two of the waypoints are placed in predetermined positions: one is placed below the runway which sets the glideslope intercept position on the runway, and the second waypoint is placed at the missed approach point. These are waypoints 2 and 1, respectively, in figure 1.

The planning algorithm connects straight lines between the waypoints with tangential circular arcs to "cut the corner" at each waypoint. The points of tangency between the turns and straight segments are denoted as gates. Each gate is associated with an airspeed, altitude, position, and nominal heading.

This information is stored on-board in the form of a "flight path" segments list, which includes turn radius (the value of 0 is used to indicate a straight segment), latitude, longitude, altitude, airspeed, heading and distance-to-go of the initial point of each segment.

The vehicle also carries on-board a complete waypoint definition list as well as a range boundary definition (described later).

Interception Algorithm

The on-board trajectory intercept planning is performed in real-time at the point that autoreturn is engaged (nominally due to loss-of-command-uplink, but possibly at the remote pilot's command). The algorithm uses an on-board Global Positioning System (GPS) for position information and then constructs an intercept trajectory from the present position, heading, and airspeed to one of the preplanned waypoints.

Only waypoints outside of the runway landing pattern (e.g., waypoints 5 and higher in figure 1) are considered as intercept targets.

The intercept algorithm considers both left- and right-hand turns from the present position (using the nominal bank angle) with tangent lines from each initial arc to a corresponding intercept arc (both left- and right-hand turns) at each candidate waypoint as indicated in figure 2. This turning onto a prescribed path is similar to that discussed in reference 1 but different in that here the goal is to intercept a predefined trajectory at a particular point. The intercept arc is expected to be flown at the scheduled waypoint speed. With the speeds and nominal bank angle defined the turn radii fall out as

$$R = \frac{v^2}{g \tan \phi_{nom}} \tag{1}$$

where v is velocity, g is the acceleration of gravity and ϕ_{nom} is the nominal bank angle. With the turn radii calculated, the turning circles can be constructed using the current and waypoint headings for orienting the axes as shown in figure 3 for the two right-turn-first approaches (left turns are the reverse).

The required parameter for defining the turn sequences is the course heading from one turning circle to the next and that is defined as

$$\Psi = \beta - k\gamma \tag{2}$$

where k is defined as –1 in the case of opposite turns, i.e. a right turn followed by a left turn or left followed by a right turn. β as depicted in the figure is the angle from the current track to a line connecting the center of the initial turn to the center of final turn. γ is also shown in figure 3 and is computed as

$$\gamma = j \sin^{-1}\left(\frac{R_2 - kR_1}{D} \right) \tag{3}$$

where k is as defined earlier, R_1 and R_2 are the turn radii, D is distance between the turn centers, and j is a parameter that has a value of +1 if the initial turn is to the right and –1 if it is to the left.

In the case of similar turns (i.e. the candidate intercept trajectory contains two left turns or two right turns) if $D < | R_1 - R_2|$ there is no solution because one turning circle lies completely inside the other and there is no way to construct a tangent line. If the turns are to be of opposite direction and $D < (R_1 + R_2)$, the circles intersect and the tangent line can not be drawn such that a change in direction arises.

Each candidate intercept trajectory is then evaluated to check if a range boundary is crossed during the maneuver: the trajectories that cross a range boundary are ignored. Of the remaining trajectories, the one that results in the shortest overall path length to the landing pattern is selected.

If no viable path to any of the outer waypoints is found, the vehicle could adjust its speed to a nominal autoreturn value, v_{nom}, and turn parallel to the closest range boundary and continue evaluating candidate trajectories to waypoints along the preplanned path until a viable choice is found. This is equivalent to finding ones way out of a maze by keeping one hand on the wall while always moving forward. This was the strategy used in the real-time simulations in this study.

An alternative strategy, which in most cases, is a faster way out of the maze takes advantage of the boundary corner points and the knowledge of their numbering sequence. To properly define the boundary, the corner points must be ordered, such that the order implies the connectivity. And, the boundary must be closed, so the first point is connected with the last. If the first boundary point is intentionally placed near the landing pattern, a strategy to adopt is to plan a path to the geographical midpoint of the highest and lowest numbered corner points to which the vehicle has line of sight. (Line of sight exists when a line from the current position of the vehicle to the corner in question intersects no boundary segments.) This ensures the vehicle will be heading back to the runway area. As more corner points come into line of sight, the target midpoint is recomputed and a corresponding path is found. If there is no path to that midpoint the algorithm temporarily backs off from using the highest and lowest numbered boundary point until a path is found.

Figure 4 shows a flight path of a vehicle implementing this line of sight strategy. For this case, as depicted in the figure, most waypoints have been removed and the range boundary artificially restricted. The boundary corner numbering has also been shown for clarity. The aircraft's initial position is near corner 7 when autoreturn is engaged. In this case there is no viable two-turn intercept trajectory to any predefined waypoint. The strategy then calls for a trajectory toward the midpoint of the highest and lowest numbered corner points to which the aircraft has line of sight. In this case that is the midpoint of corners 7 and 4 respectively (marked by the letter A in the figure). As the vehicle flies at v_{nom} toward that midpoint it continuously searches for an intercept path to any available waypoint and also checks for line of sight to the boundary corners. When the vehicle reaches the point where the second vehicle silhouette is shown, corner 3 is within line of sight so a new midpoint is computed between 7 and 3 (shown as location B in figure 4). The process continues with new midpoints computed until, while on the way to midpoint E, a viable intercept path to the waypoint at the entrance to the landing pattern is found.

The trajectories computed to fly towards a midpoint are true S-turns with no straight leg segment between the turns. This minimizes the deviation between the trajectory and a line of sight to the target midpoint and thus reduces the likelihood of a trajectory intersecting the boundary.` Trajectories that do intersect the boundary are discarded. Figure 5 depicts the geometry used to determine the lengths of the turn arcs to be commanded to come to the new course. The first arc is twice the magnitude of γ (the bearing to the end point of the first arc) and the second is the desired track minus the track at the time of the completion of the first arc. From the figure it can be seen that γ is given by

$$\gamma = \beta - j\delta \tag{4}$$

where j is as before and β is the desired change in course. The computation of δ is as follows:

$$\delta = \tan^{-1}\left(\frac{L_2 + L_1}{L_4} \right) \tag{5}$$

where,

$$L_1 = R_1 \sin\alpha \tag{6}$$

$$L_2 = (R_2 - L_1)\left|\frac{R_1}{R_2 + R_1}\right| \tag{7}$$

$$L_3 = \sqrt{R_1^2 - L_2^2} \tag{8}$$

$$L_4 = L_3 + R_1 \cos\alpha \tag{9}$$

$$\alpha = j\beta - 90^\circ \tag{10}$$

Boundary Detection Algorithm

There are many ways to describe range boundaries, but a convenient manner is to define an arbitrary region in two dimensional (2-D) space with a series of boundary points which are connected by straight segments (analogous to fence posts and fence rails defining the perimeter of a field). Given this representation of the allowed flight range, two pieces of information are of particular interest: is the vehicle still inside the bounded region and how far can it travel in any direction before leaving the range?

Interior or exterior?

Displaying the current aircraft position and overlaying the range bounds on a monitor quickly gives a pilot/operator a qualitative sense of the aircraft's position relative to the boundaries. To computationally determine whether the current or a future vehicle position (hereafter called the "test point") is inside or outside a bounded region, an area comparison can be performed in which the area of the bounded region is compared to the area of the same region with a slight modification. The modification is to insert the vehicle's present position into the list of "fence posts" and see if the area enclosed by the modified region is larger or smaller than the unmodified region, as shown in figure 6. If the area of the modified region is less than the known area of the bounded region, the vehicle or test point is inside the bounded region.

The area of an arbitrarily complex region as defined here can be computed by summing the individual areas of triangles formed by each boundary segment (fence rail) and an arbitrary fixed point (vertex). This is easily visualized for convex shapes: imagine wedges of a pie. For regions that include concave boundaries, the strategy will work if the triangular area computation yields an opposite signed area whenever the triangle encloses space outside the boundary. This may seem as though it involves a complicated bookkeeping scheme, but the area computation can be succinctly performed with a determinant in a summation loop.

$$Area = \frac{1}{2}\sum_{i=2}^{n}\begin{vmatrix} x_1 & x_i & x_{i-1} \\ y_1 & y_i & y_{i-1} \\ 1 & 1 & 1 \end{vmatrix} \tag{11}$$

Here x_i and y_i are the coordinates of the *i*th boundary point, x_1 and y_1 are the coordinates of the arbitrary vertex, and there are n boundary points.

The sign of the bounded area will depend on the direction of travel around the perimeter of the boundary, but the magnitude will be correct in the units of the coordinates squared. Note that the direction of travel around the individual triangle conveniently reverses whenever space outside the boundary is included. This method is simple to implement (the determinant, with its row of ones, can be written explicitly as five additions and three multiplications) and requires only a method to identify the boundary line that passes closest to the current position to construct the modified region.

This technique for area computation requires that the boundary be described in terms of a Cartesian coordinate system. The spherical coordinates of latitude, longitude, and altitude (including Earth radius) must be converted. This requires the selection of a local reference point from which a delta in latitude can converted into a North-South distance and a delta in longitude can be put in terms of East-West distance based on the distance per degree at the current latitude.

Distance to boundary?

Computing the range to each boundary segment involves knowing the length of the segment and the length from the test point to the segment endpoints. The question then is whether the test point is adjacent to the segment such that a perpendicular from the segment could intersect the test point. To determine this, the following inequality is evaluated:

$$\left| r_i^2 - r_{i+1}^2 \right| > d^2 \tag{12}$$

Here r_i and r_{i+1} are the distances from the test point to the segment endpoints and d is the length of the segment, as shown in figure 7. If this inequality is false, then the range to the segment is the length of the perpendicular which can be quickly found using law of cosines for finding the angle between the boundary and one of the sides connecting test point to endpoint. Then the perpendicular distance is product of the sine of the angle just computed and the distance from the endpoint to test point. Otherwise, the shortest distance to the segment is the distance to the nearer endpoint.

Course-line distance to boundary

To compute the course-line distance to any boundary, the bearing (β) from the vehicle to all the boundary points must be available. The procedure amounts to looping through n+1 boundary points (where point number n+1 = point number 1 to close the boundary) searching for a sign change in the difference between the course and $\beta(i)$ from one node to the next. Where there is a sign change, the course-line crosses the boundary segment, as shown in figure 8. For non-convex boundaries there may be more than one crossing. Solving for the intersection points is a matter of constructing equations for the boundary line and the course-line and then solving for the intersection with additional checks to be certain that the intersection is on the finite segment length and forward of the vehicle (not behind). Note that special logic must be used to account for North-South running boundary segments or courses where, when converted to a Cartesian system the slope is infinite. Multiple intersections require computing the range to each and selecting the nearest.

Turn Limiting

When maneuvering a vehicle within a confined range one must take into consideration the vehicle's turn radius when operating near the boundary. Whether a planned turn will intersect a boundary can be determined in a fashion similar to the course-line algorithm. If the distances to each segment are already known from earlier calculation and the vehicle turn radius is known then all segments further away than twice the turn radius can be excluded from consideration. The remaining segments must be checked on a case by case basis. Knowing the intended turn direction, turn radius, and vehicle position and heading; the location of the center of the turn is easily computed. With the turn center and radius known an equation for the turn circle can be constructed. The circle equation and an equation for the line segment being evaluated can be solved simultaneously to yield any intersection points, as illustrated in figure 9. The coupled equation is a quadratic and easily solved. Valid intersections will be real and on the finite segment length. In the case of two intersections on the segment the one with the smaller bearing from the current heading is the limiting case and the limiting heading change is twice the bearing angle.

Guidance Law

The guidance law associated with the autoreturn function is shown in figure 10. From a calculation of lateral error (y) and lateral error rate (ydot), provided by a navigation subroutine, an intercept angle back to the course line is formed and limited (presently to 45 degrees). This is added to the current scheduled track angle (χ_{nav}, from the flight path list) to form a track angle command, χ_{cmd}.

The current track angle, $\chi_{a/c}$, is compared to the track angle command to form the track angle error. This error is used to form a heading change to correct the error.

The heading change is added to present heading to form a heading command and this is provided to the heading command autopilot to steer the vehicle back to the desired path. This algorithm is used for both straight and turning flight path segments.

Go-around Feature

Once past waypoint 3 (figure 1), the final approach waypoint, the autoreturn function monitors airspeed, bank angle, and lateral and vertical deviations from the desired final approach path with increasingly tight error bounds. If an error bound is exceeded and sufficient fuel remains, the vehicle is commanded to climb and accelerate to pattern altitude and speed while tracking runway centerline. Once past waypoint 1, a turn to intercept the downwind leg is commanded, and the landing is attempted again.

Implementation

A Microsoft® Excel spreadsheet is used to design the waypoint and boundary lists. A Mathworks® Matlab M-script takes the waypoint list and generates the flight path list, consisting of alternating straight and circular arc segments to follow the desired path.

A prototype of the intercept-generating algorithm and boundary checking routine were written in Matlab M-scripts and later rewritten in ANSI C for use in an engineering simulation. The total implementation is approximately 1,000 lines of executable C code operating on a Silicon Graphics® workstation.

The Matlab M-script for intercept generation has been incorporated in a Matlab batch simulation for the development and evaluation of the boundary corner averaging algorithm. The batch simulation fully exercises the on-board algorithms described herein.

Results

Shown in figure 11 is a runway, an arbitrary convex range boundary, the preplanned flight path segments, and the vehicle position at the time of engagement of the autoreturn function. Also shown is the actual path flown by the vehicle (in a real-time simulation).

Figures 12 and 13 depict autoreturn intercept solutions selected by the algorithm in the simulation from a variety of initial positions and headings. Note the positions that start outside the boundary - no action is taken until the boundary is crossed, when a turn to intercept the nominal flight path is generated. Note also the large radius of the initial turn, resulting from a relatively high-speed initial condition.

Concluding Remarks

This paper has described a feasible solution to intercepting and following a preplanned flight path to return a remotely-piloted vehicle autonomously to a recovery area while avoiding prescribed arbitrary range boundaries.

At present, the use of Matlab's Stateflow product is being evaluated as another solution for the flight control system implementation of this algorithm.

Reference

1. Chandler, P.R.; Rasmussen, S.; Patcher, M.: "UAV Cooperative Path Planning", AIAA GNC, Denver, Co. August 14-17, 2000.

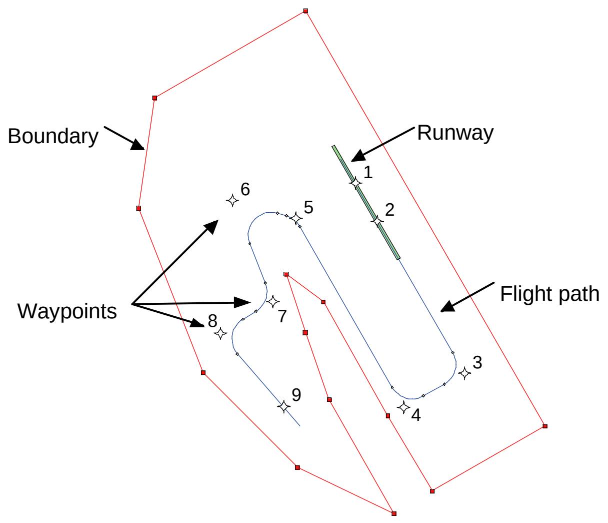


Figure 1. Autoreturn preplanned path with range boundary and waypoints shown.

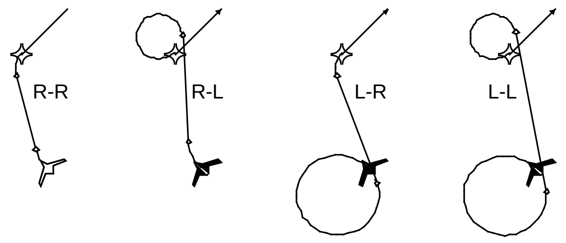


Figure 2. Candidate intercept trajectories to a waypoint.

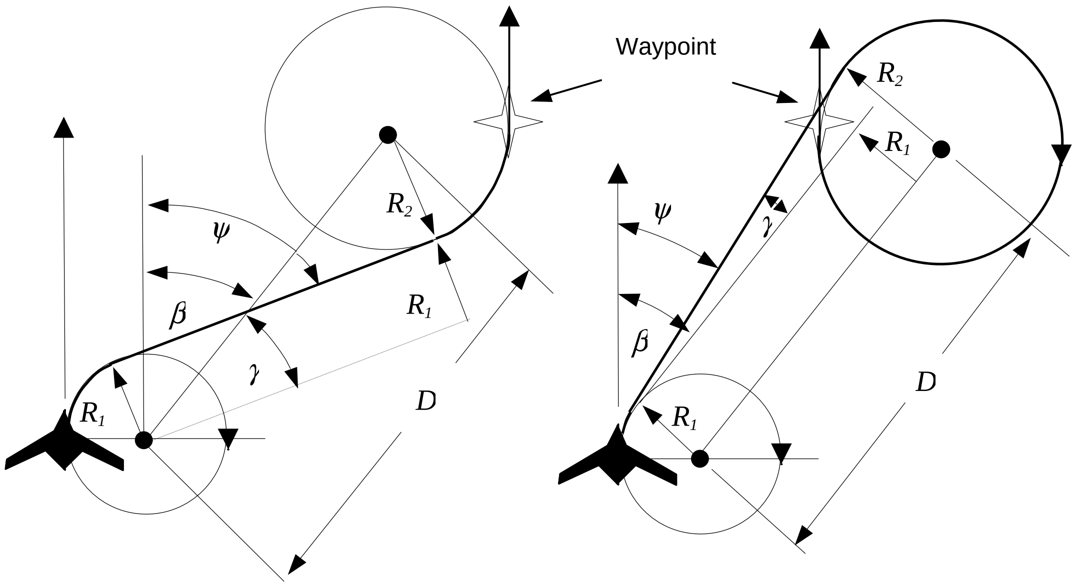


Figure 3. Geometry of waypoint intercept via right turn trajectories.

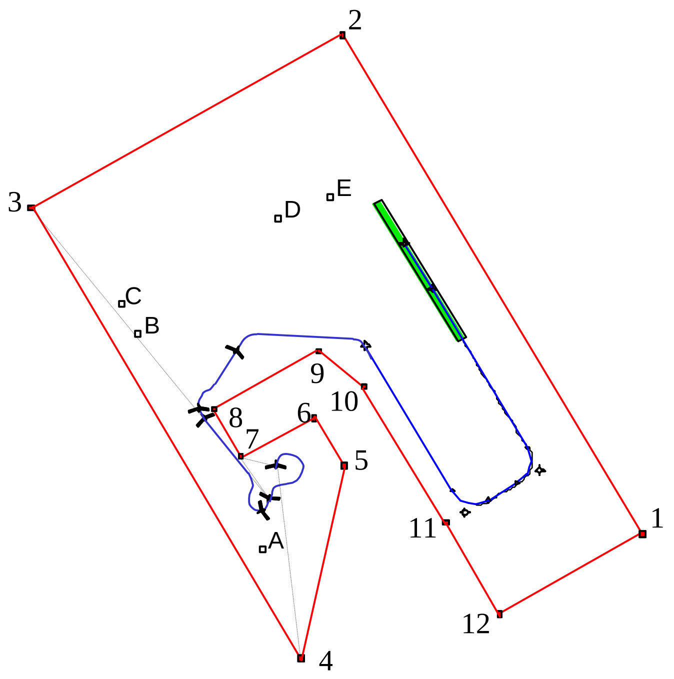


Figure 4. Flight path of vehicle within a bounded range guided by boundary corner point averaging.

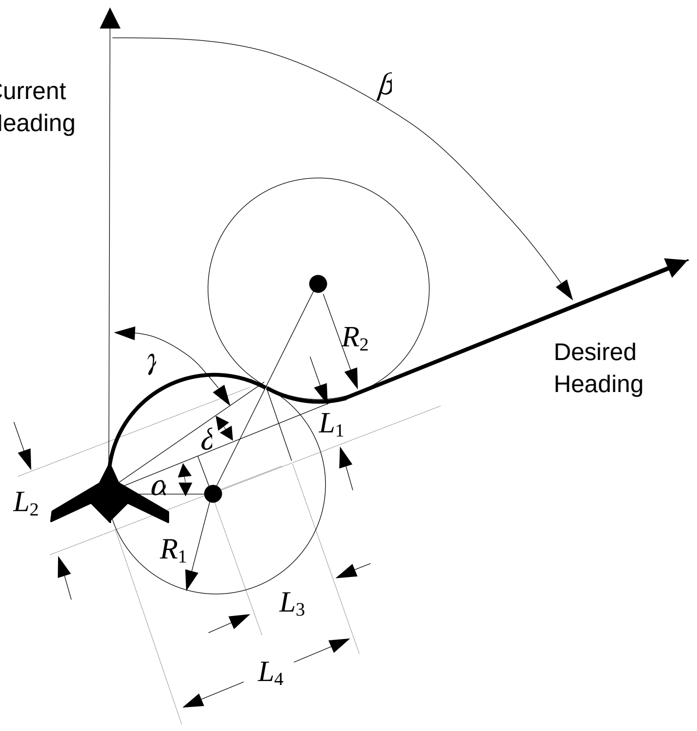


Figure 5. Geometry of right hand S-turn from current heading onto the desired heading.

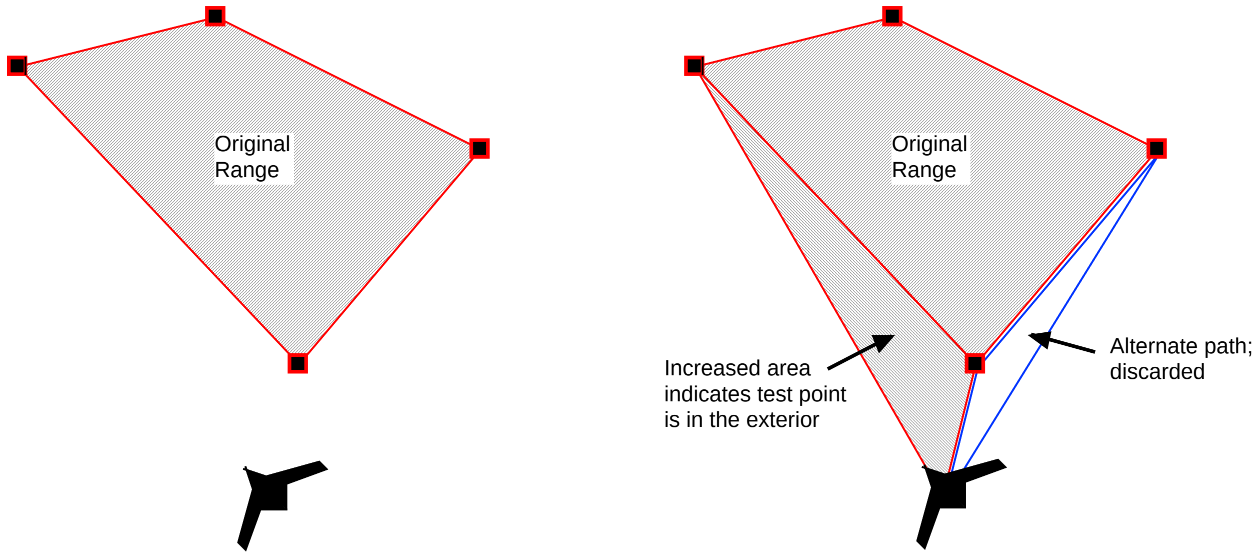

Original Range
Original Range
Increased area indicates test point is in the exterior
Alternate path; discarded

Figure 6. Test to determine interior/exterior status of the aircraft relative to the bounded range.

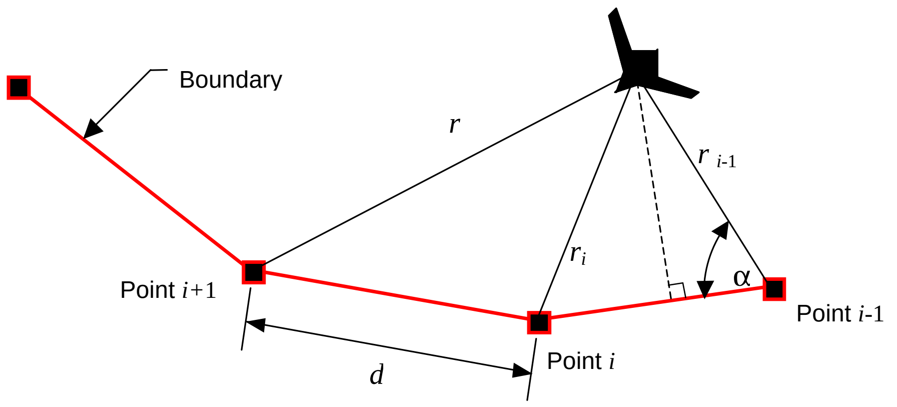


Figure 7. Geometrical layout for determining the minimum range to a boundary segment.

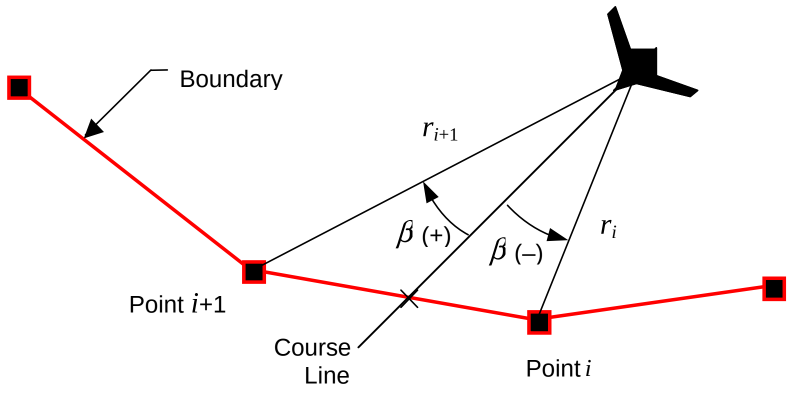


Figure 8. Schematic for determining course-line intersection with the boundary.

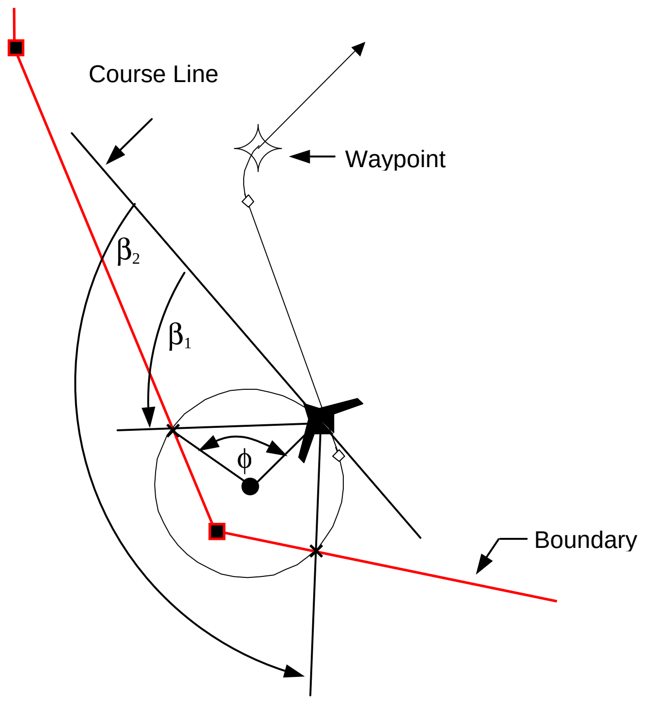
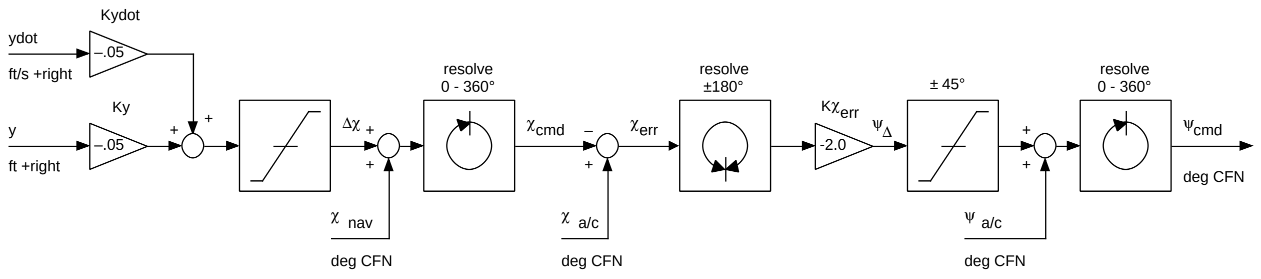


Figure 9. Depiction of the two boundary intersections that arise when considering turning flight.



CFN = Clockwise from North

Figure 10. Lateral guidance law for trajectory following autopilot.

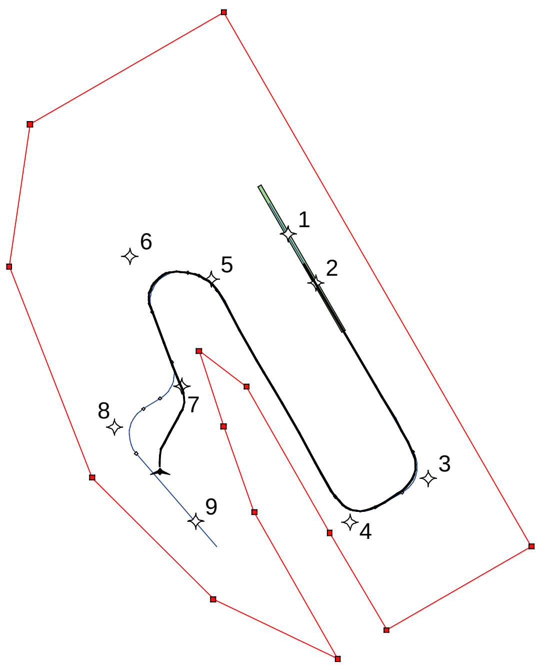


Figure 11. Nominal autoreturn trajectory from real-time simulation showing fight path intercept and following.

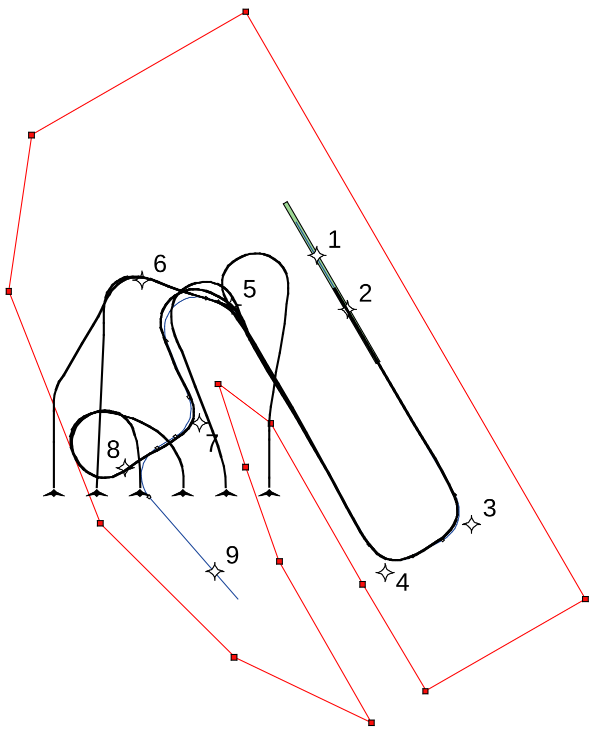


Figure 12. Autoreturn trajectories from several different initial positions.

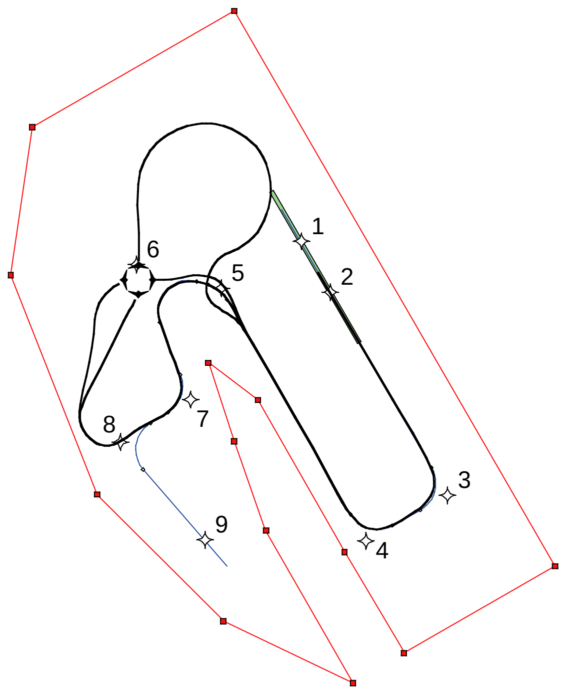


Figure 13. Autoreutrn trajectories form several different initial headings.